Einführung in C

von Henning Schwanbeck

Technische Universität Ilmenau
2003

Inhaltsverzeichnis

1 Einführung

1.1 Verfügbare Zeichen

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9			
+	-	*	/	=	\	%	&	#	!	?	^	”
#		&	<	>	(	)	[	]	{	}	:	
		|										
;	.	,	_	(Leerzeichen)								

1.2 Nichterlaubte Zeichen

ä ö ü Ä Ö Ü ß

1.3 Operatoren

In der folgenden Tabelle sind die Operatoren kurz beschrieben und auch nach deren Vorrang gruppiert.

Operator	Bedeutung	Beispiel
( )	Klammern	`durchschnitt=( x + y ) / 2`
[ ]	eckige Klammern	`array[0]`
- >	Strukturmitglied (Zeiger)	`rekord -> naechster`
.	Strukturmitglied (normal)	`rekord.name`
++	plus eins	`i++` gleicht `i = i + 1`
–	minus eins	`i- -` gleicht `i = i - 1`
(Datentyp)	Typumwandlung	`(int) gleitkommazahl`
*	Inhalt von	`*zeiger`
&	Adresse von	`zeiger = &variable` (*zeiger* zeigt auf *variable*)
-	Minus-Vorzeichen	`x = -8`
sizeof	erforderliche Speicherplatz	`sizeof(short int)`
*	Multiplikation	`produkt = zahl_1 * zahl_2`
/	Division	`cosinus = ankathete / hypotenuse`
%	Restwert einer Division	`a = 345 % 300` $\Longrightarrow$ a=45 `a = 44 % 2` $\Longrightarrow$ a=0
+	Addition	`a = x + y`
-	Subtraktion	`a = x - y`

>	größer als (ergibt Boolean Wert)	`if ( zahl > max ){...}`
> =	größer oder gleich (ergibt Boolean Wert)	`while ( zahl >= 0 ){...}`
<	kleiner als (ergibt Boolean Wert)	`while ( i < n ){...}`
<=	kleiner oder gleich (ergibt Boolean Wert)	`if ( a =<0 ){...}`
==	logisches Gleich (ergibt Boolean Wert)	`if (x==0){...}`
!=	logisches Nicht (ergibt Boolean Wert)	`if (y!=0){...}`
&&	logisches Und (ergibt Boolean Wert)	`if ( (y>=0) && (y<=9) ){...}`
||	Logisches Oder (ergibt Boolean Wert)	`if ( (a=='y') || (a=='Y') ){...}`
? :	Voraussetzung	`flag = ( i < 0 ) ? 0 : 100` $\Longrightarrow$ *flag* übernimmt den Wert von 0 wenn i < 0, und 100 wenn nicht
=	Zuweisung	a=3
+=	Addition	`i += 3` gleicht `i = i + 3`
-=	Subtraktion	`i -= 5` gleicht `i = i - 5`
=	Multiplication	`k *=6` gleicht `k = 6 * k`
/=	Division	`k /= 2` gleicht `k = k / 2`
%=	Restwert einer Division	`j %= 60` gleicht `j = j % 60`

1.4 Variablen

Variablenamen werden aus den Kleinbuchstaben, den Zahlen und dem Zeichen _ zusammengestellt. Es ist jedoch üblich daß Zahlen nur am Ende des Variablennamens stehen, z.B. `zeichen_1`.

1.5 Datentypen

Folgende Datentypen stehen zur Verfügung:

Datentyp	Beschreibung	Bemerkungen
int	ganze Zahl	es gibt auch short int, long int, unsigned int, daher unsigned short int usw.
char	Zeichen	
float	Gleitkommazahl	enthält ein Komma oder einen Exponenten
double	Gleitkommazahl mit doppelter Genauigkeit	kann mehr Ziffern oder einen größeren Exponent enthalten

1.6 Reservierte Wörter

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

2 Der Aufbau eines C- Programms

Ein C- Programm kann in folgende Einheiten unterteilt werden:

- Anweisungen
- Funktionen
- Bibliotheksfunktionen
- Variablendeklarationen
- Blöcke

2.1 Ein einfaches Beispiel

Der nachfolgende Quelltext stellt ein einfaches C- Programm dar, welches die Zeichenkette *Hello World* auf dem Bilschirm ausgibt:

```
/* Das ist ein Kommentar */

#include<stdio.h>
/* die header-Datei stdio.h wird angerufen */

int main(void)
{
    printf("Hello, world\n"); /* \n ergibt eine neue Zeile */
}
```

Ein Kommentar wird zwischen /* und */ gesetzt. Die Funktion *printf* übernimmt die Ausgabe von Zeichen, Wörtern oder Variableninhalten auf dem Bildschirm. Die geschweiften Klammern kennzeichnen allgemein gesehen einen zusammenhängenden Block und hier im Speziellen das Anfang und das Ende des Hauptprogramms. Mit dem Bezeichner *main()* wird der Beginn des Hauptprogramms gekennzeichnet.

2.2 Noch ein Beispiel

In diesem Programm wird zusätzlich zu der gewohnten Bildschirmausgabe mit dem *printf*- Kommando der Vert einer Variablen von der Tastatur unter Verwendung der *scanf*- Funktion eingelesen:

```
#include<stdio.h>

int main(void)
{
    float x, y, result;
/* x, y, result werden als Gleitkommazahlen definiert */
```

```
    printf("Geben Sie bitte zwei Zahlen ein!\n");

    printf("Zahl 1: "); scanf("%f" ,&x);
    printf("Zahl 1= %f\n",x);

    printf("Zahl 2: "); scanf("%f" ,&y);
    printf("Zahl 2= %f\n",y);

    result=x+y;
    printf("Die Summe der beiden Zahlen ist %.3f\n", result);
}
```

In C werden die einzelnen Anweisungen mit einem Semikolon getrennt.

2.3 Die Ausgabe mit printf

Das Kommando *printf* ermöglicht die Ausgabe von Zeichen, Wörtern und Variableninhalten auf Bildschirm. Durch spezielle Kontrollzeichen kann eine Formatierung der Bildschirmausgabe vorgenommen werden. So wird mit dem Steuerzeichen / der Ausgabekursor auf den Anfang der nächsten Zeile gesetzt. Weitere Steuerzeichen sind in der nachfolgenden Tabelle aufgeführt:

Steuerzeichen	Aktion
`\n`	Zeilenende (newline)
`\t`	horizontaler Tabulator
`\v`	vertikaler Tabulator
`\r`	Zeilenrücklauf
`\\`	Backslashzeichen
`\"`	Anführungszeichen
`\a`	Gong

Für die Ausgabe von Argumenten bzw. von Variablen ist die Angabe eines sogenannten Formatelementes zwingend erforderlich. In der nachfolgenden Tabelle sind diese Formatelemente aufgeführt:

Formatzeichen	Variablentyp	Beispiel
`%d`	int	3
`%f`	float	-56.4023
`%e`	float	-5.6+e01
`%E`	float	-5.6+E01

Zusätzlich kann bei der Ausgabe von Gleitkommazahlen die Anzahl der Vor- und Nachkommastellen festgelegt werden:

Formatzeichen	Variablentyp	Beispiel
`%3d`	int	`13`
`%f`	float	`-56.4023`
`%5.2f`	float	`56.34`
`%5.0f`	float	`56`
`%.1f`	float	`56.3`
`%-6d`	float	`56.3`

Das Minuszeichen ermöglicht eine linksbündige Ausgabe.

2.4 Die Eingabe mit scanf

Die Funktion *scanf* liest eine Zahl von der Tastatur ein und speichert sie unter der angegebenen Variable ab. Die Funktion *scanf* ist in der *header*- Datei *stdio.h* deklariert. Bei *scanf* ist zu beachten, daß die Adressen der Variablen übergeben werden! Es muß nach dem Prozentzeichen ein Formatzeichen angegeben werden:

Zeichen	Variablentyp
% d	integer
% f	float
% lf	double
% c	char
% s	string

Nachfolgend ein einfaches Programm zum Einlesen des Wertes der Variablen x:

```
#include<stdio.h>
int main(void)
{
    float x;
    printf("Geben Sie bitte x ein: "); scanf("%f" ,&x);
    printf("x= %f\n",x);
}
```

Die Eingabe mit `scanf` stellt eine formatierte Eingabe von Zahlen oder Zeichen dar und ist in der Anwendung nicht sehr flexibel. Im nachfolgenden Abschnitt wird daher auf die zeichenweise Eingabe mit `gets` eingegangen.

2.5 Die Eingabe mit gets

Für das Einlesen von Tatstatureingaben aller Art sollte immer das `gets`- Kommando verwendet werden. `gets` liest alle Zeichen von der Tastatur (standard input) bis zum *newline*-Zeichen - also die Enter Taste.
Nach der Betätigung der Entertaste wird das *newline*-Zeichen verworfen und ein Nullzeichen als Stringende- Markierung angefügt. Nachfolgende Abbildung verdeutlicht den Aufbau einer Zeichenkette:

Zeichen	e	s		r	e	g	n	e	t	\0
Index	0	1	2	3	4	5	6	7	8	9
Position	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.

Im nachfolgenden Programm wird ein String mit `gets` eingelesen und mit `printf` auf dem Bildschirm ausgegeben:

```
#include<stdio.h>

int main(void)
{
  char help[30];

  printf("Bitte geben Sie Ihren Vornamen ein: %s");

  gets(help);

  printf("Ihr Vorname ist: %s\n",help);
  puts(help); /*  was macht puts ? */
}
```

Das Kommando `puts` schreibt die angegebene Variable auf die Standardausgabe - den Bildschirm. Mit `puts` kann nur eine Variable ausgegeben werden.

Bei der `gets`- Funktion kann die Größe des Eingabepuffers nicht begrenzt werden. Infolgedessen kann ein Überlauf des Eingabepuffers auftreten. Als alternative Funktion kann und *sollte* das Kommando `fgets` verwendet werden:

```
#include<stdio.h>
#define max 5     /* <<<<<< maximale Stringlaenge = 5 */

int main(void)
{
  char help[max];

  printf("Bitte geben Sie Ihren Vornamen ein: ");

  fgets(help,max,stdin);

  printf("Ihr Vorname ist: %s\n",help);
  puts(help); /*  puts ist einfach !? */
}
```

Im Funktionsaufruf `fgets(help,max,stdin)` ist mit `stdin` das Standardeingabegerät - also die Tastatur angegeben. Der Parameter bzw. Konstante `max` gibt die Anzahl der einzulesenden Zeichen inklusiv dem Nullzeichen an. Alle nachfolgend eingegebenen Zeichen werden ignoriert.

Im nächsten Abschnitt wird unter Verwendung der `gets`- Funktion eine Gleitkommazahl eingelesen.

2.6 Die Umwandlung von Zeichen in eine Zahl

Wenn mit Hilfe der `gets`- Funktion als Ergebnis eine Zahl erwünscht ist, dann muß diese Zeichenkette in eine Zahl umgewandelt werden.

2.6.1 Die Umwandlung einer Zeichenkette in eine Gleitkommazahl

Mit Hilfe der Funktion `strtod` kann eine Zeichenkette in eine Gleitkommazahl umgewandelt werden. Im nachfolgenden Beispiel entspricht die eingegebene Zeichenkette `0.173e+1` einer Körpergröße von `1.73` :

Zeichen	0	.	1	7	3	e	+	0	1	\0
Index	0	1	2	3	4	5	6	7	8	9
Position	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.

Dazu der passende Quelltext:

```
#include<stdio.h>
#include<stdlib.h>
#define max 30     /* <<<<<< maximale Stringlaenge = 30 */

int main(void)
{
  double x;
  char help[max];
  char *endptr;

  printf("Bitte geben Sie Ihre Koerpergroesse ein: ");

  gets(help);    /* <<<<<< fgets() waere besser ! */
                 /* fgets(help,max,stdin);        */
  x=strtod(help,&endptr);

  printf("Ihre Koerpergroesse betraegt: %lf\n",x);

  if (*endptr!=0) printf("Der Rest von help ist : %s\n",endptr);
}
```

Eine Zeichenkette kann nur dann in eine Gleitkommazahl umgewandelt werden, wenn diese Zeichenkette erlaubte Zeichen (`0..9 - + E e .`) enthält.
Wenn in der Zeichenkette nichterlaubte Zeichen wie zum Beispiel ein Komma `,` statt eines Punktes `.` dann wird die Zeichenkette nicht konvertiert.

In diesem Falle zeigt der *Pointer* `endptr` auf die Position des ersten nichterlaubten Zeichens im der eingelesenen Zeichenkette.

Die Funktion `strtod` überprüft den angegebenen String auf Leerzeichen und Vorzeichen und wandelt dann den String in eine Gleitkommazahl um.

2.6.2 Die Umwandlung einer Zeichenkette in eine ganzzahlige Variable

Mit Hilfe der Funktion `strtol` kann eine Zeichenkette in eine ganzzahlige Gröβe umgewandelt werden:

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
  int x;
  char help[30];
  char *endptr;

  printf("Bitte geben Sie Ihr Alter ein: ");

  gets(help);

  x=strtol(help,&endptr,10);

  printf("Ihr Alter betraegt: %d\n",x);

  if (*endptr!=0) printf("Der Rest von help ist : %s\n",endptr);
}
```

Der Zeiger `endptr` zeigt nach der Konvertierung auf die *fehlerhafte* Stelle bei der Konvertierung. Die Zahl `10` im Funktionskopf gibt die Basis der Umwandlung an.

Die Funktion `strtol` überprüft den angegebenen String auf Leerzeichen und Vorzeichen und wandelt dann den String in eine ganzzahlige Größe um.

2.7 Aufgaben

1. Schreiben Sie ein Programm, welches eine einzugebende Gleitkommazahl halbiert und das Ergebnis auf dem Bildschirm darstellt!

2. Multiplizieren Sie zwei einzugebende ganze Zahlen miteinander und stellen Sie das Ergebnis auf dem Bildschirm dar!

3. Berechnen Sie die mittlere Geschwindigkeit eines Fahrzeuges (Input: Weg und Zeit; Output: Geschwindigkeit)!

4. Berechnen Sie unter Verwendung des Pythagoras-Satzes die Diagonale eines Rechtecks (Input: Seitenlänge a und b; Output: Umfang, Fläche und Diagonale)!

5. Berechnen Sie das Volumen einer Kugel (Input: Kugelradius; Output: Kugelvolumen)! (Kugelvolumen: $V_K = \frac{4}{3} \cdot \pi \cdot r^3$)

6. Berechnen Sie die kinetische Energie eines Fahrzeuges mit der Geschwindigkeit v (Input: v, Output: kinetische Energie E_{kin})

7. Berechnen Sie die Gewichtskraft als Sonderfall der Gravitationskraft $F_G = m \cdot g$ (Input: Masse m, Output: Kraft F_G)!

8. Üben Sie an den vorangegangen Aufgaben die Ausgabeformatierung mit dem *printf* Kommando!

3 Bedingungsanweisungen

Die *if-else-* Anweisung wertet eine Bedingung oder einen Vergleich aus und führt dann entsprechende Anweisungen aus. Ein Vergleich wird mit Hilfe von Vergleichsoperatoren durchgeführt:

Operator	Bezeichnung	Beispiel
<	kleiner als	x<3
>	größer als	x>4.6
<=	kleiner oder gleich	x<=y
>=	größer oder gleich	a>=-0.05
!=	ungleich	i!=k
==	gleich	i==k

Dabei darf der Zuweisungsoperator = nicht mit dem Vergleichoperator == verwechselt werden!

3.1 Die einfache Bedingungsanweisung

Bei einer einfachen Bedingunganweisung ist der optionale *else-* Zweig nicht erforderlich. Es werden die Anweisungen ausgeführt, wenn die Bedingung wahr ist. Ist das Ergebnis der Bedingung falsch, wird der Anweisungsblock übersprungen und nicht ausgeführt.

```
if(Bedingung)
{
   Anweisung_1;
   .
   .
}
```

Das nachfolgende Programbeispiel zeigt einige Beispiele für die einfache Bedingunganweisung:

```
#include<stdio.h>
#DEFINE pi 3.14

main()        /* die Funktion main wird definiert */
{
  float x=5, y=-4;
  if(x<3)    { printf("Richtig!\n");}
  if(x>y)    { printf("Feierabend!\n");}
  if(x<=-11) { printf("Keine Lust!\n");}
  if(x!=y)   { printf("Was soll denn dass?\n");}
  if(x==5)   { printf("Ist das wirklich gleich?\n");}
  if(x<7)    printf("So geht es aber nur bei einer Anweisung!\n");
}
```

Wenn es nur eine Anweisung gibt, kann man die geschweiften Klammern weglassen. Wie aus diesen Beispielen ersichtlich, ist die *else*-Anweisung optional und kann weggelassen werden.

3.2 Die vollständige if- else- Anweisung

Ist bei der *if-else-* Anweisung die Bedingung wahr, werden die Anweisungen in dem nach der Bedingung folgenden Block ausgeführt - hier `Anweisung_1`. Ist das Ergebnis der Bedingung falsch, werden die Anweisungen in dem nach *else* folgenden Block ausgeführt - hier `Anweisung_2`.

```
if(Bedingung)
{
   Anweisung_1;
}
else
{
   Anweisung_2;
}
```

Ein Programmbeispiel verdeutlich die *if-else-* Anweisung:

```
#include<stdio.h>
#define pi 3.14
main()        /* die Funktion main wird definiert */
{
  float x=5, y=-4;
  if(x<3)    { printf("Richtig !\n");} else {printf("Dann eben nicht!\n");}

  if(x>pi) { printf("Feierabend !\n");}
      else { printf("Jetzt ist es mit Pi klar!\n");}

 /* Bei einer Anweisung geht es ohne die geschweiften Klammern */
  if(x>7)  printf("So geht es auch!\n");
      else printf("wenn man es weiss!\n");
}
```

In den meisten Fällen ist es für die Erhöhung der Übersichtlichkeit günstiger, den Anweisungsteil in geschweiften Klammern einzuschließen!

3.3 Die geschachtelte if-else- Anweisung

Da der *else-* Zweig einer *if-else-* Anweisung optional ist, entsteht eine Mehrdeutigkeit, wenn ein *else-* Zweig in einer verschachtelten Folge von *if-else-* Anweisungen fehlt. Dem wird dadurch begegnet, daß der *else-* Zweig immer mit dem letzten *if* verbunden wird, für das noch kein *else-* Zweig existiert.

```
#include<stdio.h>
#define pi 3.14
main()        /* die Funktion main wird definiert */
{
  float x=5, y=4;

  if(x>pi)
    if (x>y)
      printf("Stimmt genau!\n");
  else
    printf("Zu welchem if gehoert dieser else Zweig?\n");
}
```

Bei solcherarts verschachtelten Anweisungen ist es generell günstiger, mit Blöcken zu arbeiten, daß heißt, entsprechend zusätzliche geschweifte Klammern zu setzen:

```
#include<stdio.h>
#define pi 3.14
main()        /* die Funktion main wird definiert */
{
  float x=5, y=4;

  if(x>pi)
  {
    if (x>y)
      printf("Stimmt genau!\n");
  }
  else
    printf("Jetzt ist es eindeutig!\n");
}
```

3.4 Die Mehrfachverzweigung

Für eine Mehrfachentscheidung, d. h. eine Auswahl unter mehreren Alternativen kann auch die *switch*- Anweisung verwendet werden, wenn die Alternativen ganzzahligen (ordinalen) Werten eines Ausdrucks vom Typ *integer* entsprechen:

```
#include<stdio.h>
#define pi 3.14

main()        /* die Funktion main wird definiert */
{
int karte=9;

switch(karte)
```

```
{
       case 7: case 8: case 9: case 10:
            printf("%d" , karte );
            break;
       case 1:
            printf("As");
            break;
       default : /* Dieser Zweig ist optional! */
            printf("andere Gesichtskarte");
            break;
}
```

Der *default*-Fall tritt dann auf, wenn keiner der anderen Fällen auftritt. Eine wichtige Bedingung für die *Switch*- Anweisung ist, daß alle *case*- Marken unterschiedlich sein müssen. Wenn eine *case*- Marke gefunden wurde, werden die Anweisungen ausgeführt und mit Hilfe des *break*- Kommandos an das Ende der *switch*- Anweiung gesprungen.

3.5 Übungsaufgaben

1. Berechnen Sie unter Verwendung des Pythagoras-Satzes die Diagonale eines Rechtecks (Input: Seitenlänge a und b; Output: Umfang, Fläche und Diagonale)! Testen Sie die eingelesenen Seitenlängen unter Verwendung der *if-else*- Anweisung auf positive oder negative Werte und lösen Sie entsprechende Aktionen aus (z. B. Fehlermeldung)!

2. Bestimmen Sie von drei einzugebenen ganzen Zahlen mit Hilfe der *if-else*- Anweisung die größte Zahl!

3. Sortieren Sie drei ganze ganze Zahlen in aufsteigender Folge der Größe nach unter Verwendung der *if-else*- Anweisung!

4 Wiederholungsanweisungen

4.1 for-Schleifen

Bei einer `for`- Anweisung ist die Anzahl der Wiederholungen durch die Vorgabe der Parameter Anfang, Schrittweite und Ende bekannt. Im nachfolgenden Programm wird bei jedem Durchlaufen der `for`- Schleife der Wert der Laufvariablen `i` auf der Konsole ausgegeben:

```
#include<stdio.h>
int main(void)
{
  int i,n;
n=10;
for (i=0;i<n;i=i+1)    printf( "%d " , i );
}
```

Die Wiederholungsanweisung wird also *zehnmal* ausgeführt. Im nächsten Programm iwird die Schrittweite verdoppelt, so daß der Anweisungsteil nur noch *fünfmal* durchlaufen wird:

```
#include<stdio.h>
int main(void)
{
  int i,n;
n=10;
for (i=0;i<n;i=i+2)
  {  /* Beginn des Anweisungsteils */
    printf( "%d " , i );
  }  /* Ende des Anweisungsteils */
}
```

Die Parameter einer `for`- Anweisung sollten im Anweisungsteil nicht geändert werden! Die Anzahl der Wiederholungen bei einer Schrittweite von `1` ergibt sich in den oben aufgeführten Programmbeispielen aus der Differenz von Anfangswert und Endwert (+1!). Die Laufvariable - hier `i`- der Anfangswert und der Endwert müssen vom gleichen ordinalen Variablentyp sein. Die Standardschrittweite bei der `for`- Anweisung ist immer 1. Der Wert der Laufvariablen ändert sich damit bei jedem Durchlaufen der Schleife um den Wert 1 oder entsprechend der vorgebenen Schrittweite.

Im nachfolgenden Programmbespiel wird mit einer `for`- Anweisung die Summe der Zahlen von `0` bis `9` berechnet:

```
#include<stdio.h>
int main(void)
{
  int i,s;
```

```
s=0;
for (i=0;i<10;i=i+1)
  {
    s=s+i;
  }
printf( "Summe: %d\n " ,s );
}
```

Das nächste Programmbespiel berechnet die Summe von 100 Zufallszahlen:

```
#include<stdio.h>
#include<math.h>
int main(void)
{
  int i,s;
s=0;
for (i=0;i<100;i=i+1)
  {
    s=s+drand48();
    /* drand48 erzeugt Zufallszahl in 0..1 */
  }
printf( "Summe: %d\n " ,s );
}
```

4.2 while-Schleifen

Eine *while*-Schleife führt die entsprechenden Befehle durch, solange die *while*-Bedingung erfüllt (wahr) ist.

```
#include<stdio.h>
#define pi 3.14
main()
{
  int zahl=0, summe=0;

  printf("Geben Sie Zahlen ein! Beenden Sie die Eingabe mit -1!\n");
  while( zahl != -1 )
  {
    scanf( "%d" , &zahl ) ;
    summe = summe +zahl ;/* summe += zahl */
  }
}
```

Die Bedingung wird am Anfang der Schleife geprüft (Der Pessimist!). Daher muß jede Variable, die geprüft wird, schon am Anfang der Schleife einen Wert beinhalten. In dem

oben angegebenen Beispiel wird die Schleife überhaupt nicht durchgeführt wenn man als erste Zahl die -1 eingibt. Werden mehrere Anweisungen ausgeführt, dann müssen diese in einem Block zusammengefaßt werden. Oftmals wird solch eine Wiederholungsanweisung mit einem sogenannten Zähler oder Laufindex verknüpft:

```
#include<stdio.h>
#define pi 3.14
main()
{
  int zahl=0, summe=0, index;

  index=0;
  printf("Geben Sie 5 Zahlen ein! \n");
  while( index<5 )
  {
    index=index+1;
    printf("%d. Zahl: ",index);
    scanf( "%d" , &zahl ) ;
    summe = summe +zahl ;/* summe += zahl */
  }

  printf("Summe= %d\n",summe);
```

4.3 do-while-Schleifen

```
summe=0;
zahl=0;
do
{
    scanf("%d", &zahl);
    summe+=zahl;
}while(zahl!=0);
```

Bei einer *do-while* Schleife wird die Bedingung erst am Ende der Schleife geprüft. Daher wird die Schleife mindestens einmal durchgeführt.

4.4 Aufgaben

1. Geben Sie eine beliebige Anzahl von Zahlen vom Typ *float* ein und berechnen Sie deren Summe mit der do-while-Anweisung! Die Eingabe wird durch 0 beendet!

2. Geben Sie eine ganze Zahl *f* mit *f>0* ein (und berechnen Sie die Fakultät *f!*

   a) mit der *while*-Anweisung!

   b) mit der *for*-Anweisung!

c) mit der *do-while*-Anweisung!

3. Bestimmen Sie unter Verwendung der *for*-Anweisung und der *if*-Anweisung

   a) das Maximum von zehn einzugebenden Zahlen!

   b) das Maximum von n Zufallszahlen! (Hinweis: Eine Zufallszahl wird mit der Funktion rand( ) erzeugt. Um diese Funktion verwenden zu können, müssen Sie die *header*-Datei *stdlib.h* in das Programm einbinden!)

   c) das Minimum von n einzugebenden Zahlen!

4. Geben Sie folgende Zahlen unter Verwendung der *for*-Anweisung auf dem Bildschirm aus!

   a) alle ganze Zahlen zwischen (einschließlich) 0 - 1000

   b) die geraden Zahlen

   c) die ungeraden Zahlen

   d) alle Zahlen zwischen (einschließlich) 0 - 1000 die sich durch 2, 3 oder 5 teilen lassen

5. Lassen Sie ein Zeichen eingeben und geben Sie folgendes auf dem Bildschirm aus: "rot" , wenn das Zeichen 'r' oder 'R' beträgt, " grün" , wenn es 'g' oder 'G' beträgt und "schwarz" für sonstige Zeichen. Benutzen Sie dazu die *switch*-Anweisung!

5 Arrays

5.1 Eindimensionale Arrays (Vektoren

Ein Array dient dazu, einen Satz von Variablen des *gleichen* Typ zusammenzufassen. Die einzelnen Elemente eines Arrays werden *direkt* über den Index angesprochen.

Die mathematische Darstellung eines Vektors lautet: $\vec{a} = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{Bmatrix}$

Die Deklaration eines Vektors, dessen Elemente hier in diesem Beispiel Variablen vom Typ *int* sind, erfolgt mit :

```
int feld1[100] ;
```

Damit enthält das eindimensionale Array mit dem Namen *feld1* 100 Elemente, d. h. die Größe des Arrays beträgt 100:

```
array_name[0]
array_name[1]
.
.
array_name[99]
```

Der Index eines Array beginnt bei *0* und läuft bis *n-1*, in diesem Beispiel bis *99*!

Das Belegen eines Feldelementes mit einem Zahlenwert erfolgt dann mit:

```
array_name[0] = 5 ;
```

Das Belegen aller Feldelemente mit Inhalten (Zahlenwerten) erfolgt dann mit:

```
array_name[]={ 5 , 7 , 3 , 2 , 1 , 2 , 36 , 42 , 73 }
```

Unter Verwendung einer Wiederholungsanweisung kann man in einfacher Form auf alle Elemente eines Vektor zugreifen:

```
#include<stdio.h>

int main(void)
{

int  index;      /* Deklaration des Laufindizes */
float otto[100]; /* Deklaration des Vektors */
float summe=0;

printf("Summe von 10 Zahlen\n");
```

```
printf("==================\n");

for( index=0 ; index<10 ; index++ )
{
  printf("Geben Sie die %d. Zahl ein: ",index+1);
  scanf( "%f" , &otto[index]);
  summe=summe+otto[index];
}

printf("Die Summe von 10 Zahlen ist %f\n",summe);
} /* Schluss des Hauptprogramms */
```

5.2 Zweidimensionale Felder (Matrizen)

Man betrachtet ein eindimensionales Array als Liste, daher kann man ein zweidimensionales Array als Tabelle betrachten, ein dreidimensionales Array als Satz von Tabellen, usw. In diesem Abschnitt wird auf die Deklaration und die Arbeit von umd mit zweidimensionalen Feldern - hier als Matrizen bezeichnet - eingegangen. Eine Matrix stellt sich in allgemeiner mathematischer Schreibweise so dar: $\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$

und in nachfolgender Gleichung sind alle Zeilen- und Spaltenindizes aufgeführt:

$$|a| = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{77} & a_{76} & a_{77} & a_{78} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{78} & a_{86} & a_{87} & a_{88} \end{vmatrix}$$

Die Deklaration einer Matrix wird wie folgt vorgenommen:

```
float auto[25][37];
int  fahrrad[14][1834];
```

In dieser Deklaration wird eine Matrix, bestehend aus 25 Zeilen und 37 Spalten, definiert. Jedes Element dieser Matrix stellt eine Variable vom Typ *float* dar. Mehrdimensionale Arrays werden wie folgt mit Werten belegt:

```
int values [2][5]={ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 }
/* oder auch */
int values[2][5]={ { 1 , 2 , 3 , 4, 5 } , { 6 , 7 , 8 , 9 , 10 } }
```

Das Array namens *values* kann man als Tabelle bestehend aus 2 Zeilen und 5 Elementen (Spalten) betrachten. Die Reihenfolge eines solchen Arrays ist wie folgt:
values[0][0]
values[0][1]
values[0][2]
values[0][3]
values[0][4]
values[1][0]
values[1][1]
values[1][2]
values[1][3]
values[1][4]

Also steigt der letzte Index zuerst, der erste zuletzt.

Das nachfolgende Programmbeispiel zeigt mit Hilfe zweier ineinander verschachtelten Wiederholungsanweisungen die Belgung aller Elemente einer Matrix mit Zufallszahlen unter Verwendung der *rand()*- Funktion:

```
#include<stdio.h>
#include<math.h>
#define zeile 111
#define spalte 111

int main(void)
{

int  i,j,anzahl;      /* Deklaration der Laufindizes */
float otto[zeile][spalte]; /* Deklaration der Matrix */
float summe=0;

printf("Summe von aller Matrixelemente\n");
printf("====================\n");

for( i=0 ; i<zeile ; i++ )
{
  for( j=0 ; j<spalte ; j++ )
  {
    otto[i][j]=rand();
    summe=summe+otto[i][j];
  }
}
anzahl=zeile*spalte;
printf("Die Summe von %d Zahlen ist %f\n",anzahl,summe);
```

```
} /* Schluss des Hauptprogramms */
```

5.3 Aufgaben

1. Deklarieren Sie einen Vektor mit 3000 Elementen von Typ float und belegen Sie jedes Element mit einer Zufallszahl!
2. Berechnen Sie die Summe über alle Feldelemente unter Verwendung der for-Anweisung! Achten Sie auf der Feldgrenze!
3. Bestimmen Sie die Summe aller Feldelemente mit geradem Index!
4. Deklarieren Sie eine Matrix mit einer Struktur von 100 Zeilen und 30 Spalten vom Typ double, und belegen Sie jedes Matrixelement mit dem Wert 0! Belegen Sie jedes Element mit einer Zufallszahl!
5. Berechnen Sie die Summe über alle Matrixelemente!
6. Geben Sie die Summe jeder Spalte einzeln aus!
7. Berechnen Sie die Summe der Hauptdiagonalelemente!
8. Bestimmen Sie das Maximum in jeder Zeile der Matrix!
9. Bestimmen Sie das Maximum in der Hauptdiagonale der Matrix!

6 Zeichen und Zeichenketten

6.1 Zeichen

6.1.1 Einleitung

Bei der Deklaration eines Zeichens mit `char zeichen_name;` wird für diese Variable einen Speicherplatz von einem Byte im Hauptspeicher (RAM) reserviert.

Jede Zeichenkonstante hat einen ganzzahligen Zahlenwert, welcher durch den Zeichensatz des Rechners bestimmt ist. Die eindeutige Zuordnung eines Zeichens zu einem Zahlenwert ist im sogenannten ASCII- Zeichensatz festgelegt.

6.1.2 Deklaration

Die Deklaration eines Zeichens erfolgt mit

```
char zei1, buchstabe ;
```

6.1.3 Das Arbeiten mit Zeichen

Im nachfolgenden Programmbeispiel wird der Inhalt eines Zeichens von der Tastatur eingelesen und anschließend auf dem Bildschirm ausgegeben:

```
#include<stdio.h>

int main(void)
{
  char zei1;

  printf("Geben Sie ein Zeichen ein: ");

  scanf( "%c" , &zei1 ) ;

  printf("Das eingetippte Zeichen war: %c\n",zei1);
}
```

Das folgende Programm schreibt die Kleinbuchstaben von a bis y auf den Bildschirm.

```
#include<stdio.h>

int main(void)
{
  char zeichen;
  int cc;

  for ( zeichen ='a' ; zeichen<'z' ; zeichen++ )
  {
```

```
    cc=(char)zeichen;
    printf("%c %d\n",zeichen,cc);
    printf(" ");
  }

printf("War das schwer !\n");
}
```

6.1.4 Übungsaufgaben

1. Schreiben Sie ein Programm, welches die Zeichen von k bis y unabhängig vom Zeichensatz des Rechners ausgibt.

2. Schreiben Sie ein Programm, welches den Dezimalwert der Zeichen von S bis X ausgibt.

3. Schreiben Sie ein Programm, welches das gesamte Alphabet in Großbuchstaben ausgibt.

4. Schreiben Sie ein Programm, welches einen Buchstabe einliesst und den wieder als Großbuchstabe ausgibt. Dabei wird eine sinnvolle Fehlermeldung erwartet.

6.2 Zeichenketten

6.2.1 Einleitung

Zeichenketten (strings) stellen in C formal betrachtet ein Array von Zeichen dar. Eine konstante Zeichenkette wird vom Compiler intern als ein Vektor von Zeichen dargestellt.Dabei wird am Schluß (Stringende) ein zusätzliches Zeichen, das Zeichen `\0` - auch Nullzeichen genannt - angehängt, um das Stringende zu kennzeichnen.

Die Stringverarbeitungsfunktionen benötigen dieses Zeichen, damit sie das Stringende erkennen. Deshalb muß bei der Speicherung von Zeichenketten stets ein Speicherplatz für das Nullzeichen vorgesehen werden.

6.2.2 Deklaration

Die Deklaration einer Zeichenkette erfolgt mit:

```
char baumart[20];
```

6.2.3 Das Arbeiten mit Zeichenketten

Die Ein- und Ausgabe von Zeichenketten ist im nachfolgenden Programm in einfacher Form dargestellt:

```
#include<stdio.h>

int main(void)
{
  char baumart[30];

  printf("Welchen Baum sehen Sie gerade? ");
  scanf("%s" , baumart);  /* <<<<<<<<<<<<<< */
  printf("Sind Sie sicher das es eine %s war?\n" , baumart);
}
```

Die Funktion `scanf` liest bis zum ersten Leerzeichen im eingebenen String! Um die Eingabe bis zum Zeilenende inclusiv aller Leerzeichen zu lesen benutzt man besser die Funktion `gets`:

```
#include<stdio.h>

int main(void)
{
  char baumart[30];

  printf("Welchen Baum sehen Sie gerade? ");
  gets(baumart);   /* <<<<<<<<<<<<<< */
  printf("Sind Sie sicher das es eine %s war?\n" , baumart);
}
```

Eine Zeichenkette wird, wie schon in der Einleitung erwähnt, mit dem Null-Zeichen `\0` beendet.

Zeichen	L	i	n	d	e	\0
Index	0	1	2	3	4	5
Position	1.	2.	3.	4.	5.	6.

In die oben vereinbarte Zeichenkette passen also nur 29 normale Zeichen - das 30. Zeichen muß daß Nullzeichen sein.
Es gibt verschiedene Funktionen, Zeichenketten zu manipulieren. All diese Funktionen sind in der header Datei *string.h* aufgeführt. Nachfolgend eine Auswahl einiger Stringoperationen:

6.2.3.1 Länge einer Zeichenkette

Bei der Bestimmung der Länge einer Zeichenkette wird das Nullzeichen nicht mitgezählt!

```
#include<stdio.h>
#include<string.h>

int main(void)
```

```
{
  char baumart[30];
  int laenge;
  printf("Welchen Baum sehen Sie gerade? ");
  scanf("%s" , baumart);
  printf("Sind Sie sicher das es eine %s war?\n" , baumart);

  laenge=strlen(baumart);
  printf("Der String besteht aus %d Zeichen\n",laenge);
}
```

6.2.3.2 Verbinden von Zeichenketten Die Funktion *strcat* verbindet die beiden Strings s2 und s1. Das Stringendezeichen \0 vom String s1 wird dabei vom String s2 überschrieben.

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char s1[30] = "Rechen";
  char s2[30] = "zentrum";

  printf("s1= %s \n" , s1);
  printf("s2= %s \n" , s2);

  strcat( s1 , s2 );
  printf("Ergebnis: %s\n" , s1);
}
```

6.2.3.3 Kopieren von Zeichenketten Die Funktion *strcpy(ziel,quelle)* kopiert den String namens *quelle* auf den String namens *ziel*:

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char ziel[30] = "Heute";
  char quelle[30] = "Morgen";

  printf("Ziel vor strcpy= %s \n" , ziel);

  strcpy( ziel , quelle );
```

```
  printf("Ziel nach strcpy= %s \n" , ziel);
}
```

6.2.3.4 Vergleichen von Zeichenketten Die Funktion *strcmp(s1,s2* führt einen *zeichenweisen* Vergleich der beiden Strings durch. Die beiden Strings werden solange verglichen, bis ein Zeichen unterschiedlich oder bis das Stringendezeichen `\0` erreicht ist:

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char s1[30] = "A";
  char s2[30] = "B";
  int result;

  printf("Vergleich von zwei Strings\n");
  printf("==========================\n");

  result=strcmp(s1,s2);

  if (result<0) printf("s1 kleiner als s2\n");
  if (result==0) printf("s1 = s2\n");
  if (result>0) printf("s1 groesser als s2\n");
}
```

Beim Vergleich werden die entsprechenden Zeichen voneinander subtrahiert. Sobald das Ergebnis der Subtraktion eine Zahl ungleich 0 ist, wird der Vergleich abgebrochen. Sind die beiden Strings ungleich lang und bis zum letzten Zeichen des kürzeren Strings gleich, so wird das entsprechende Zeichendes längeren Strings mit dem Stringende `\0` des kürzeren Strings verglichen.

Die Funktion *strncmp(s1,s2,n);* führt einen Zeichenweisen Vergleich der Strings s1 und s2 durch. Abweichend von *strcmp* erfolgt nach n Zeichen ein Abbruch des Vergleiches.

Bei dem nachfolgendem Beispiel werden die Strings `s1` und `s2` von der Tastatur eingelesen:

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char s1[30];
  char s2[30];
  int result;
```

```
  printf("Vergleich von zwei Strings\n");
  printf("==========================\n");
  printf("s1:"); gets(s1);
  printf("s2:"); gets(s2);
  result=strcmp(s1,s2);

  if (result<0) printf("s1 kleiner als s2\n");
  if (result==0) printf("s1 = s2\n");
  if (result>0) printf("s1 groesser als s2\n");
}
```

6.2.3.5 Vertauschen des ersten und letzten Zeichens

Im nachfolgenden Programm werden vom eigebenen String das erste und das letzte Zeichen vertauscht:

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char help,s1[30];
  int i,laenge;

  printf("s1:"); gets(s1);


  laenge=strlen(s1);

  printf("Vorwaerts: ");
  for (i=0;i<laenge;i++)
  {
    printf("%c",s1[i]);
  }
  printf("\n");

  printf("Rueckwaerts: ");
  for (i=laenge;i>-1;i--)
  {
    printf("%c",s1[i]);
  }

  printf("\nVertauschung erstes und letztes Zeichen: ");
  help=s1[0];
```

```
  s1[0]=s1[laenge-1];
  s1[laenge-1]=help;
  printf("%s\n",s1);
}
```

6.2.4 Übungsaufgaben

1. Schreiben Sie ein Programm, welches die Länge des eingebenen Strings berechnet und auf dem Bildschirm ausgibt!

2. Geben Sie den Inhalt des in Aufgabe 1 eingegebenen Strings in umgekehrter Reihenfolge auf dem Bildschirm aus!

3. Schreiben Sie ein Programm, welches den Inhalt von vier Zeichenketten einliest und miteinander verbindet! Arbeiten Sie mit der *strcat*-Funktion! Aus wieviel Zeichen besteht der neue String?

4. Lesen Sie eine Zeichenkette mit einer Länge von mindestens 10 Zeichen ein und vertauschen Sie das erste und letzte Zeichen!

5. Lesen Sie eine Zeichenkette ein und untersuchen Sie diese Zeichenkette (bzw. jedes Element dieser Zeichenkette) auf die Existenz des Zeichens e oder i oder anderer Zeichen!

7 Blöcke und Funktionen

7.1 Was ist ein Block?

Ein *Block* ist eine Folge von Anweisungen, die sequentiell hintereinander ausgeführt werden.

Die Anweisungen eines Blockes werden durch sogenannte *Blockbegrenzer* zusammengefaßt. In C werden als Blockbegrenzer die geschweiften Klammern verwendet. Ein Block - im Allgemeinen bestehend aus Vereinbarungsteil und Anweisungsteil - stellt sich im Programm folgendermaßen dar:

```
{  /* Beginn des Blockes */
   int    i,j,k;   /* Deklarationsteil  */

   for (i=0;i<100;i++)  /* Anweisungen */
   {
     printf("i= %d\n",i);
   }
}  /* Ende des Blockes */
```

Ein Block kann aus mehreren verschachtelten Blöcken (innere und äußere Blöcke) bestehen. Die in einem Block definierten Variablen sind auch nur innerhalb dieses Blockesgültig (sichtbar). Die in den äußeren Blöcken als globale Variablen deklarierten Variablen sind auch in den inneren Blöcken sichtbar.

7.2 Funktionen

Prozeduren und Funktionen (Blöcke) sind Mittel zur Strukturierung von Programmen und dienen damit der logischen Unterteilung des Programmes (Quelltextes). Funktionen sind Unterprogramme, die mehrmals aufgerufen werden können und daher die Mehrfachverwendung von Programmtext ermöglichen.

7.3 Funktionen ohne Parameter

Dieses Programm enthält eine parameterlose Funktion, d.h. es werden keine Variableninhalte vom Haupprogramm an die Funktion namens *fehler* gegeben.

```
#include<stdio.h>

void fehler()
{
  printf("Division durch 0 nicht moeglich!\n");
  return;
}
```

```
int main(void)
{
  float zahl,reziproke;

  printf("Geben Sie bitte eine Zahl ein!");
  scanf("%f", &zahl);

  if(zahl==0)     fehler();
  else {
         reziproke=1/zahl;
         printf("Die Reziproke ist %f\n" , reziproke);
       }
}
```

Die in unserem erstem Programmbeispiel verwendete Funktion namens *fehler* gibt nur eine Meldung auf dem Bildschirm aus. Sie benötigt keine Eingangsparameter und liefert auch kein Ergebnis (Rückgabewert). Wird kein Parameter vom Hauptprogramm an die Funktion und umgekehrt übergeben, so ist die Syntax des Funktionskopfes:

```
void fehler()
```

Der Bezeichner *void* bedeutet also, dass von der Funktion keine Daten an das Hauptprogramm zurückgegeben werden.

Gibt die Funktion einen Wert zurück, so kann (aber muss nicht!) er abgeholt werden:

```
printf("Das Ergebnis ist: %f\n",quadrat() );
```

oder

```
float help;
help=quadrat();
printf("Das Ergebnis ist: %f\n",help);
```

7.4 Funktionen mit Parametern

7.4.1 Input- und Outputparameter einer Funktion

Oftmals ist es notwendig, eine gegebene Funktion aufzurufen und dieser Funktion als Input- Parameter den Inhalt einer oder mehrerer Variablen anzugeben. Vor dem Erstellen einer Funktion muß also der Programierer exakt die Schnittstellen dieser Funktion bezüglich Input- und Outputparameter definieren: Wenn von einen Rechteck mit den gegebenen Seitenlängen `a` und `b` der Umfang und der Flächeninhalt berechnet werden soll, dann sind folgende Variablen als Input- und Outputparameter definiert:

- Inputparameter
  - Seitenlänge `a`

  - Seitenlänge `b`

- Outputparameter (Rückgabewerte)
  - Flächeninhalt `f`
  - Umfang `u`

In den nachfolgenden Abschnitten wird auf Funktionen mit unterschiedlicher Anzahl von Input- und Outputparametern eingegangen.

7.4.2 Funktionen mit einem Rückgabewert

Im nachfolgenden Programmbeispiel wird auf ein Funktion namens *wurzel* zurückgegriffen, welche als Inputparameter die Zahl erhält, von welcher die Quadratwurzel berechnet werden soll. Mit der *return-* Anweisung wird als Rückgabewert der Funktion die Quadratwurzel übergeben.

```
#include<stdio.h>
#include<math.h>

void fehler(float help) /* F. ohne Rueckgabewert */
{
  printf("Fehler: Die eingegebene Zahl ist negativ:%f\n",help);
  return;
}

float wurzel(float help) /* F. mit Rueckgabewert */
{
  float otto;

  otto=sqrt(help):
  return(otto);
}

int main(void)
{
  float zahl,ergebnis;

  printf("\nProgramm zur Wurzelberechnung\n");
  printf("Geben Sie eine Zahl ein:");
  scanf("%f" , &zahl);

  if(zahl<0) fehler(zahl);
  else {
    ergebnis=wurzel(zahl);
```

```
    printf("Die Quadratwurzel ist %.2f\n",ergebnis);
  }
}
```

7.4.3 Funktionen mit mehreren Rückgabewerten

In C ist eine sogenannte *call by reference*- Schnittstelle in der Syntax nicht vorgesehen. Man kann im Parameterkopf nicht direkt die (Output-)Variablen angeben, sondern realisiert die Übergabe von einer Funktion heraus in das Hauptprogramm mittels Zeiger. Kurz gesagt: Es wird ein Zeiger auf den aktuellen Parameter übergeben.

7.4.3.1 Ein einfaches Beispiel

```
#include<stdio.h>

void start(float *x)
{
  *x = 7.782;
  printf("x innnerhalb der Funktion; %f\n",*x);
}

int main(void)
{
  float a;

  printf("\nRueckgabewert einer Funktion\n");

  start(&a);

  printf("a= %f\n",a);
}
```

Beim Aufruf von `start(&a)` wird die lokale Variable `x` in der Funktion namens *start* deklariert. Diese Variable `x` wird innerhalb der Funktion mit einem Zahlenwert belegt: `*x = 7.782;`. In der Deklaration einer Zeigervariable kann man dann im übertragenen Sinne folgende Syntax: `float *x = &a` erkennen. Damit wird dem Objekt, auf das der Zeiger a zeigt, der Wert der Variablen `x` zugewiesen.

7.4.3.2 Die Lösung der quadratischen Gleichung Das nachfolgende Programmbeispiel erläutert den Aufbau einer Funktion, welche mehr als einen Rückgabewert (hier zwei Rückgabewerte) aufweist. Diese zwei Rückgabewerte werden über die Parameterliste der Funktion unter Verwendung von Zeigern zurückgegeben.

In diesem Beispiel wird die Lösung der Gleichung $x_{1,2} = -\frac{a}{2} \pm \sqrt{\frac{a^2}{4} - b}$ in der Funktion namens *quak* berechnet. Als Inputparamater sind die Variablen a und b vorgesehen. Die

Lösung der Gleichung wird von der Funktion an die Variablen `x1` und `x2` übergeben. Diese Variablen `x1` und `x2` können als Outputparameter bezeichnet.

```
#include<stdio.h>
#include<math.h>

float quak(float fa,float fb,float *fx1,float *fx2)
{
  float help;

  help=fa*fa/4 -fb;
  if (help<0) return(help); /* Check ob sqrt moeglich */
  *fx1=-fa/2+sqrt(help);
  *fx2=-fa/2-sqrt(help);
  return(0);
}

int main(void)
{
  float a,b,x1,x2,test;

  printf("\nProgramm zur Loesung der quadr. Gleichung\n");

  printf("Eingabe von a: ");  scanf("%f" , &a);
  printf("Eingabe von b: ");  scanf("%f" , &b);
  /* a=9 b=3 ==> x1=-0.3466 x2=-8.6533 */

  test=quak(a,b,&x1,&x2);

  if (test<0) printf("Wurzelausdruck negativ (%f)\n",test);
  else {
    printf("x1= %f:\n", x1);
    printf("x2= %f:\n", x2);
  }
}
```

Beim Aufruf der Funktion *quak* werden die *lokalen* Variablen `fx1` und `fx2` angelegt und mit dem Wert des zugehörigen Parameters initialisiert:

```
float *fx1 = &x1
float *fx2 = &x2
```

7.4.4 Rekursive Funktionen

Eine rekursive Funktion ist, einfach gesagt, eine Funktion, die sich selber aufruft. Die Verwendung eine rekursiven Funktion ist nur dann sinnvoll, wenn die zu lösende Auf-

gabenstellung rekursiven Typs ist wie zum Beispiel die Berechnung der Fakultät einer ganzen Zahl

Rekursion kann auch an Stelle einer *for*-Schleife verwendet werden. In der Praxis wird man ein so einfaches Problem nicht rekursiv lösen, aber dieses Beispiel soll Ihnen prinzipiell zeigen, wie die Rekursion funktioniert. Das Beispiel hat die selbe Ausgabe wie die folgende *for*-Schleife:

7.4.4.1 Berechnung der Fakultät mit for- Schleife

```
#include<stdio.h>

int main(void)
{
  int f,fak, i;

  printf("\nFakultaet von: ");
  scanf("%d",&fak);


  f=1;
  for (i=1;i<fak+1;i++) {
  f=f*i;
  }

  printf("Fakultaet von: %d ist %d\n",fak,f);
}
```

7.4.4.2 Berechnung der Fakultät mit einer rekursiven Funktion

```
#include<stdio.h>

int fakultaet(int nn)
{
  if (nn>=1) return nn*fakultaet(nn-1);
             else return(1);
}

int main(void)
{
  int f,fak;

  printf("\nFakultaet von: ");
  scanf("%d",&fak);
```

```
  f=fakultaet(fak);

  printf("Fakultaet von: %d ist %d\n",fak,f);
}
```

7.5 Übungsaufgaben

1. Erstellen Sie ein Programm unter Verwendung einer Funktion zur Ausgabe einer Bildschirmzeile mit 80 Gleichheitszeichen!

2. Erstellen Sie ein Programm, welches eine Funktion zur Berechnung der 3. Potenz einer Zahl enthält!

3. Erstellen Sie ein Programm unter Verwendung einer Funktion zur Berechnung der Fläche und des Umfangs eines Rechtecks! Gegeben sind die Seitenlängen `a` und `a`.

4. Erstellen Sie ein Programm unter Verwendung einer Funktion zur Berechnung des *Minimums* und *Maximums* von n Zufallszahlen!

5. Erstellen Sie ein Programm unter Verwendung einer Funktion zur Berechnung der Summe von n Zufallszahlen (Input: Anzahl n; Output Summe s)! (Input: Anzahl n; Output: min, max)!

6. Erstellen Sie ein Programm, welches eine Funktion zur Berechnung folgender Kugelparameter enthält:

   - Input: Kugelradius
   - Output: Kugelvolumen $V_K = \frac{4}{3} \cdot \pi \cdot r^3$ und Kugelfläche $A_K = 4 \cdot \pi \cdot r^2$

7. Erstellen Sie ein Programm unter Verwendung einer Funktion zur Bestimmung des Schnittpunktes von zwei Geraden!

   Input:

   a) Gerade 1: Punkt 1: x_{11}, y_{12}; Punkt 2: x_{21}, y_{22}
   b) Gerade 2: Punkt 3: x_{31}, y_{32}; Punkt 4: x_{41}, y_{42}

   Output: Schnittpunkt x_{sp}, y_{sp}

8 Arbeiten mit Dateien

Bisher befanden sich die Daten im Arbeitsspeicher (RAM) und wurden in Form von Variablen abgelegt.Um nun größere Datenbestände zu bearbeiten oder Daten auszutauschen, werden diese Variablen oder Datenbestände auf einem Datenträger (Festplatte, Diskette, ZipDrive) dauerhaft gespeichert.

Jede Datei ist eindeutig durch ihren Namen und das dazughörige Verzeichnis gekennzeichnet. Der Zugriff auf eine Datei erfolgt durch die Angabe des Dateinamens.

Der folgende Abschnitt geht vorzugsweise auf Textdateien ein.

8.1 Textdateien

Als Textdateien werden spezielle, nur sequentiell verarbeitbare Dateien bezeichnet, deren Elemente vom Typ CHAR sind.

Zum Schreiben in eine Datei oder zum Lesen aus einer Datei muß diese Datei geöffnet werden.

Der Inhalt dieser Datei kann mit einem gewöhnlichen Editor wie z. B. *notepad* (Windows95) oder dem *nedit* (Unix) eingesehen werden.

Für das Arbeiten mit Dateien wird ein zusätzlicher Variablentyp, ein sogenannter *Filepointer* eingeführt:

```
FILE *datei_zeiger;
```

Der Filepointer ist ein Zeiger auf eine Struktur, die Angaben über den Bearbeitungszustand der Datei enthält. Die Einzelheiten dieser Struktur, die in stdio.h definiert ist, sind für uns nicht wichtig.

Im nachfolgendem Programm wird eine Datei geöffnet und ewas hineingeschrieben:

```
#include<stdio.h>

#define MAX 5
#define LAENGE 10

int main(void)
{
  char quatsch[30] = "Ist das ein Wetter heute!";
  char datei_name[LAENGE];
  FILE *datei_zgr;

  printf("Dateiname: ");
  scanf("%s" , datei_name);

  datei_zgr=fopen(datei_name,"w");

  fprintf(datei_zgr , "%s \n" , quatsch);
```

```
  /* fprintf schreibt formatierte daten in die Datei */

  fputs(quatsch,datei_zgr);
  /* fputs schreibt nur den Inhalt in die Datei */
  fclose(datei_zgr);
}
```

Die möglichen Dateioperationen wie lesen, schreiben oder anhängen (append) werden durch die im *fopen* angeführten Attribute bestimmt:

```
datei_zgr=fopen(datei_name,"w");
```

w eine neue Datei nur fürs Schreiben öffnen (wenn eine solche Datei schon vorhanden ist wird sie durch eine neue ersetzt)

r eine schon vorhandene Datei (!) wird zum Lesen geöffnet

a eine schon vorhandene Datei wird geöffnet; wenn eine solche Datei nicht vorhanden ist wird eine neue erstellt

Im oben angegebenen beispielprogramm haben wir Daten in eine Datei geschrieben. Das nachfolgende Programm versucht nun diese Daten wieder auszulesen:

```
#include<stdio.h>

#define MAX 5
#define LAENGE 10

int main(void)
{
  char quatsch[30] = "Ist das ein Wetter heute!";
  char datei_name[LAENGE];
  FILE *datei_zgr;

  printf("Dateiname: ");
  scanf("%s" , datei_name);

  datei_zgr=fopen(datei_name,"r");

  fscanf(datei_zgr , "%s \n" , quatsch);
  printf("fscanf: %s\n",quatsch);
  /* fscanf liest bis zum ersten Leerzeichen */


  fgets(quatsch,10,datei_zgr);
  printf("fgets: %s\n",quatsch);
```

```
  /* fgets liest hier 10 Zeichen ein */
  fclose(datei_zgr);
}
```

8.2 Aufgaben

1. Erzeugen Sie 111 Zufallszahlen vom Typ *float* und schreiben Sie diese Zahlen in einer Datei!

2. Lesen Sie den Inhalt dieser Datei wieder ein und berechnen Sie den Mittelwert aller Zahlen!

3. Lesen Sie den Inhalt dieser Datei als ein Array von Zeichenketten ein und geben Sie ihn auf dem Bildschirm aus!

4. Deklarieren Sie ein mehrdimensionales Array vom Typ *char* (5 Namen, die jeweils 10 Zeichen lang sein dürfen). Belegen Sie diese Vektorelemente mit Inhalten und sichern Sie den Vektorinhalt in einer Datei!

5. Lesen Sie den Inhalt der in der vorherigen Aufgabe erzeugten Datei wieder ein und geben Sie ihn auf dem Bildschirm aus!

9 Strukturen

9.1 Was ist eine Struktur?

Eine Struktur ist eine Zusammenfassung von mehreren Variablen (i. A. unterschiedlichen Typs). Die einzelnen Variablen werden als *Komponenten* dieser Struktur bezeichnet. Wie schon erwähnt, können Strukturen Variablen unterschiedlichen Typs enthalten - im Gegensatz zu Array, welche grundsätzlich Variablen gleichen Typs beinhalten.

Beispiele dafür sind Datenbankeinträge in Form von Adreßeinträgen oder Lagerbeständen und in einfacher Form komplexe Zahlen bestehend aus Real- und Imaginärteil. Mit Hilfe einer Struktur werden zusammengehörige Daten zusammengefaßt.

In nachfolgender Abbildung ist eine Struktur zur Charakterisierung eines Autos dargestellt:

Typ	Farbe	Alter	Preis
string	string	integer	real

In der ersten Zeile der Tabelle sind die Parameter eines Autos in Form von Variablen angegeben und in der zweiten Tabellenzeile sind die dazugehörigen Variablentypen angegeben. Bei der Definition für eine Struktur muß für jede Komponente deren Namen und Typ angegeben werden.

9.2 Deklaration

Für die Definition eines Strukturtyps wird das Schlüsselwort `struct` verwendet. Im nachfolgenden Programmbeispiel wird die Struktur einer Adresse definiert:

```
#include<stdio.h>

int main(void)
{
  struct adresse { char strasse[40];
                   int hausnummer;
                   int plz;
                   char stadt[20];
                 };

  struct adresse adr1 = {"Poststrasse" , 17 , 98693, "Ilmenau"};

  printf("Strasse: %s\n",adr1.strasse);
  printf("Hausnummer: %d\n",adr1.hausnummer);
  printf("Postleitzahl: %d\n",adr1.plz);
  printf("Stadt: %s\n",adr1.stadt);
}
```

Die Struktur *adresse* besteht aus den vier Komponenten: `strasse, hausnummer, plz, stadt`. Die Komponenten können von einem beliebigen Typ sein - also auch selbst wieder eine

Struktur.

Auch die Komponenten eines Arrays könne vom Typ `struct` sein:

```
#include<stdio.h>

int main(void)
{
  struct adresse { char strasse[40];
                   int hausnummer;
                   int plz;
                   char stadt[20];
                 };

  struct adresse adr[5];

  char s1[20]="Helmholtzring";
  char s2[20]="Ilmenau";

  strcpy(adr[1].strasse,s1);
  strcpy(adr[1].stadt,s2);

  adr[1].hausnummer=27;
  adr[1].plz=98693;

  printf("Strasse: %s\n",adr[1].strasse);
  printf("Hausnummer: %d\n",adr[1].hausnummer);
  printf("Postleitzahl: %d\n",adr[1].plz);
  printf("Stadt: %s\n",adr[1].stadt);
}
```

Man kann auch mit dem *typedef* Bezeichner arbeiten und sich erst einen neuen Datentyp definieren:

```
#include<stdio.h>

int main(void)
{
  typedef struct { char strasse[40];
                   int hausnummer;
                   int plz;
                   char stadt[20];
                 } adresse;

  adresse adr1 = {"Poststrasse" , 17 , 98693, "Ilmenau"};
```

```
  printf("Strasse: %s\n",adr1.strasse);
  printf("Hausnummer: %d\n",adr1.hausnummer);
  printf("Postleitzahl: %d\n",adr1.plz);
  printf("Stadt: %s\n",adr1.stadt);
}
```

Es ist auch möglich, gleichzeitig den Typ einer Struktur und die Variablen dieses neuen Typs zu deklarieren:

```
struct kart_koor { float x;
                   float y;
                 } p1, p2;
```

Hier wurden die Variablen `p1` und `p2` vom Typ `kart_koor` deklariert. Weitere Variablen vom Typ `kart_koor` können später über den Typ `kart_koor` deklariert werden:

```
struct kart_koor p3,p4;
```

Der Typname `kart_koor` kann auch weggelassen werden. Das macht aber nur Sinn, wenn alle Variablen diesen Typs sofort deklariert werden, da ohne den Typnamem später keine Variablen diesen Typs vereinbart werden können.

9.3 Übungsaufgaben

1. Erzeugen Sie eine kleine Autodatenbank unter Verwendung von *struct*-Variablen mit mindestens fünf verschiedenen Autos und belegen Sie diese Struktur mit Inhalten! Ein Auto kann zum Beispiel folgende Parameter haben: Typ, Hersteller, Farbe, Preis, Alter. Definieren Sie die fünf Autos

   a) mit Hilfe von fünf verschiedenen Variablen

   b) mit Hilfe eines Arrays, welches aus fünf Elementen besteht.

2. Erzeugen Sie eine kleine Warenbestandsliste unter Verwendung von *struct*-Variablen mit mindestens 4 verschiedenen Artikeln. Diese Artikel sind jeweils durch Bezeichnung, Preis, Anzahl und Farbe charakterisiert. Speichern Sie die Warenbestandsliste in einer externen Datei!

3. Definieren Sie diese in Aufgabe 1 erzeugten *record*-Variablen als dynamische Variablen!

4. Definieren Sie sich eine zweite dynamische Variable mit der in Aufgabe 1 verwendeten Struktur und belegen Sie die einzelnen record-Elemente mit Inhalten!

10 Zeiger

10.1 Definition

Der Arbeitsspeicher (RAM) eines Rechners ist in Speicherzellen eingeteilt. Jede Speicherzelle trägt eine Nummer. Diese Nummer wird als (Speicher-) Adresse bezeichnet.

Ein Zeiger ist eine Variable, welche die Adresse eines im Speicher befindlichen Objektes aufnehmen kann.

Ein Zeiger ist ein Variablentyp, der nicht auf dem Wert eines Datenbestandes hinweist, sondern auf dessen Adresse im Speicherplatz.

10.2 Deklaration

Ein Zeiger wird wie eine Variable folgendermaßen deklariert:

```
int   *p_eule;
float *P_falke;

int alter;
float x;
```

Der Stern vor dem Variablennamen weist darauf hin, daß die Variable vom Typ Zeiger ist. Die Datentypangabe *int* weist darauf hin, daß der Zeiger auf eine Variable vom Typ *int* zeigt. Diese Zeigervariable namens `p_eule` enthält also die Adresse der *int*-Variable namens `p_eule`.

Der Wertebereich einer Zeigervariablen vom Typ *Zeiger* ist die Menge aller physikalischen Speicheradressen und dem `NULL`- Zeiger. Der Zeiger `NULL` ist ein vordefinierter Zeiger, dessen Wert sich von allen regulären Zeigern unterscheidet. Der `NULL`- Zeiger zeigt auf kein gültiges Speicherobjekt.

Mit der Definition einer Zeigervariablen wird noch kein Speicherplatz für ein Objekt des angegebenen Typs `int` oder `float` reserviert.

10.3 Operationen mit Zeigern

Man speichert die Adresse der Variable `alter` in der Zeigervariable `p_eule` wie folgt:

```
#include<stdio.h>

int main(void)
{
  int   *p_eule;
  float *p_falke;

  int alter;
  float x;
```

```
  alter=95;

  p_eule=&alter;   /*  p_eule zeigt auf die Adresse von alter*/

}
```

10.4 Dereferenzierung

Wurde einem Zeiger ein Wert zugewiesen, so möchte man natürlich auch auf das sogenannte referenzierte Objekt, also auf den Speicherinhalt zugreifen können,

```
#include<stdio.h>

int main(void)
{
  int   alpha;
  int *p_alpha;

  int *otto;

  alpha=37;
  printf("Alpha= %d\n",alpha);

  p_alpha=&alpha;
  *p_alpha=74;
  printf("Alpha= %d\n",alpha);

  *&alpha=13;
  printf("Alpha= %d\n",alpha);

  otto=p_alpha;
  printf("Alpha= %d\n",*otto);
}
```

auf den der Zeiger zeigt.

Zum Arbeiten mit Zeigern stehen die beiden Zeigeroperatoren `&` und `*` zur Verfügung:

- `&` wird als Adreßoperator bezeichnet. Mit ihm erhält man die Adresse eines Objektes (einer Variable).
- `*` wird als Inhaltsoperator oder Dereferenzierungsoperator bezeichnet. Mit ihm erhält man den Inhalt des Objektes.

Zusammenfassung

`&variable`	die Adresse der Variable
`zeiger=&variable`	*zeiger* beinhaltet die Adresse der Variable
`*zeiger`	der Datenbestand worauf *zeiger* zeigt

11 Dynamische Speicherverwaltung

11.1 Statische Variablen

Prinzipiell unterscheidet man bei jeder Programmiersprache zwischen *statischen* und *dynamischen* Variablen. Eine statische Variable ist eine Variable die in einem Programm vereinbart wird:

```
int   i;
```

Bei der Deklaration erhält die Variable einen Bezeichner oder einen Variablennamen - hier den Variablennamen `i`. Nach dieser Deklaration kann auf den Inhalt der Variablen über ihren Namen zugegriffen werden. Eine solche Variable heißt statisch, weil ihr Gültigkeitsbereich und ihre Lebensdauer durch die statische Struktur des Programmes festgelegt ist. Ebenfalls ist für diese statische Variable während der gesamten Programmlaufzeit ein bestimmter Speicherbereich reserviert.

11.2 Dynamische Variablen

Dynamische Variablen erscheinen nicht explizit im Deklarationsteil. Für eine dynamische Variable wird bei Bedarf der Speicherbereich im Hauptspeicher (RAM) reserviert. Bei Bedarf heißt, daß der Speicherbereich genau dann reserviert wird, wenn mit dieser Variablen im Programm gearbeitet wird - wenn es also z. B. notwendig ist, einen Zahlenwert im Hauptspeicher abzulegen. Wird diese Variable (Speicherbereich) nicht mehr benötigt, dann wird dieser Speicherbereich wieder freigegeben.

11.3 Zuweisung von Speicher

In der Programmiersprache C erfolgt die Reservierung von Speicherplatz mit Hilfe der `malloc()`- Funktion (memory allocation). Die Freigabe von Speicherplatz wird mit der Funktion `free()` vorgenommen. In der Programmiersprache PASCAL stehen dazu die Funktionen `new` und `dispose` zur Verfügung. Das nachfolgende Prommierbeispiel zeigt, wie der Zeiger `z1` auf den Anfang eines Speicherbereiches zeigt, in welchem der Inhalt einer Variable vom Typ `int` abgelegt wird:

```
#include<stdio.h>
#include<stdlib.h> // Diese Lib ist fuer malloc notwendig !!

int main(void)
{
  int *z1;
  int anzahl;

  z1=NULL;  // Initialisierung von z1
```

```
  anzahl=sizeof(int);
  printf("Wir brauchen %d Byte Speicher!\n",anzahl);

  z1=malloc(anzahl);
  printf("Prima! Jetzt haben wir die Bytes.\n");

  if (z1!=NULL) *z1=10;

// ueblich: if ((z1=malloc(sizeof(int)))!=NULL) *z1=10;

  printf("z1= %d\n",*z1);
  printf("Anzahl: %d\n",anzahl);
}
```

Die `malloc()`- Funktion ist in `stdlib.h` deklariert - daher ist es notwendig, diese Headerdatei mit im Programmkopf anzugeben.

11.4 Freigabe von Speicher

Die Freigabe nicht benötigten Speichers erfolgt mit der Funktion `free()`. Die freigegebene Speicherbereich kann dann erneut mit `malloc()` zugewiesen werden. Das nachfolgende Programmbeispiel zeigt die Anwendung von den Funktionen `malloc()` und `free()`:

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
  int *z1;

  z1=NULL;  // Initialisierung von z1

  z1=malloc(sizeof(int)); // Speicherzuweisung
  if (z1!=NULL) *z1=10;   // Check auf gueltige Adresse

  printf("z1= %d\n",*z1);

  free(z1);               // Speicherfreigabe
}
```

11.5 Noch ein Programmbeispiel

Dieses Beispiel zeigt die Reservierung und Freigabe von einem Speicherbereich für einen String:

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
  char *z1;
  char s1[20]="Welch ein Tag!";

  int laenge;

  laenge=sizeof(char)*strlen(s1)+1;
  printf("Anzahl der Bytes: %d\n",laenge);

  z1=malloc(laenge); // Gib mir Speicher!

  if (z1!=NULL) strcpy(z1,s1);   // Stringablage im Heap

  printf("String s1: %s\n",s1);
  printf("String z1: %s\n",z1);

  free(z1);               // Speicherfreigabe
  printf("Sollte ich es jetzt verstanden haben?\n");
}
```

11.6 Übungsaufgaben

Lösen Sie die folgenden Aufgaben unter Verwendung von dynamischen Variablen:

1. Was sind statische Variablen? Was sind dynamische Variablen? Was ist der Unterschied zwischen statischen und dynamischen Variablen?

2. Erklären Sie folgende Wörter:

   - Speicheradresse
   - Speicherinhalt
   - Offset

   Versuchen Sie diese Begriffe mit Hilfe von Beispielen zu erklären!

3. Geben Sie unter Verwendung der *while*-Anweisung den Umfang eines Kreises in Abhängigkeit vom Kreisradius im Bereich von 23,5 cm bis 43,5 cm auf dem Bildschirm aus! Der Kreisradius wird bei jedem Durchlauf um 0,5 cm erhöht. Reservieren Sie für jede Variable mit Hile der `malloc()`- Funktion Speicherplatz und geben Sie diesen weider mit Hilfe der `free()`- Funktion wieder frei!

4. Deklarieren Sie einen Vektor mit 3000 Elementen vom Typ float und belegen Sie jedes Element mit einer Zufallszahl (*rand*-Funktion) !

5. Bestimmen Sie die Summe aller Feldelemente mit geradem Index unter Verwendung der `%` -Funktion!

6. Deklarieren Sie eine Matrix mit einer Struktur von 100 Zeilen und 30 Spalten vom Typ float und initialisieren Sie jedes Element mit dem Wert 0 ! Belegen Sie jedes Matrixelement mit einer Zufallzahl (*rand*-Funktion)!

7. Berechnen Sie die Summe über alle Matrixelemente unter Verwendung der

   - *for*
   - *do-while*
   - *while*

   Anweisung ! Achten Sie auf die Feldgrenzen !

8. Berechnen Sie die Summe der Hauptdiagonalelemente !

12 Listen

12.1 Einführung

Die bisherige Methode, einen Satz von Variablen des gleichen Formats zu speichern, war das Array. Die neue Methode, die Liste, hat zwei Vorteile:

1. Die Variable wird Speicherplatz dynamisch, d.h. während des Programmablaufes und nicht während der Deklaration zugewiesen; daher kann man Listenelemente während des Programmablaufes hinzufügen oder löschen.

2. Die Verknüpfung der einzelnen Listenelemente erfolgt durch Zeiger. Listeelemente ändern.

Jedes Listeelement stellt eine Struktur dar:

```
 struct adresse {
                    char name[20];
                    char strasse[20];
                    int hnummer;
                    long int plz;
                    struct adresse *naechster;
                  };
```

Jede Struktur enthält zusätzlich zu den eigentlichen Informationen einen Zeiger, welcher auf die Anfangsadresse des nächsten Elements (Struktur) verweist:
`struct adresse *naechster;`.

Ein Element einer Liste (eine Struktur) wird wie folgt deklariert:

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
struct adresse {
                  char name[20];
                  char strasse[20];
                  int hnummer;
                  char plz[5]
                  listen_el *naechster
                };
struct adresse *adr1
 }
```

Bis jetzt wurde zu jedem Listenelement direkt über seinen Namen zugegriffen. Mit einer (verknüpften) Liste, kann jedes Listenelement über den Zeiger im vorherigen Listenelement, und dieser Zeiger wiederum über den vorherigen Zeiger, erreicht werden. Der

Anfang einer Liste ist durch einen Zeiger auf das erste Listenelement gekennzeichnet. Der Speicher für ein neues Listenelement muß mit der `malloc()`- Funktion angefordert werden.

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
  struct adresse {
                   char name[20];
                   char strasse[20];
                   int hnummer;
                   long int plz;
                   struct adresse *naechster;
                 };

  struct adresse *adr1, *anfang, *vor, *position, *last;

  char s1[20];
  char *endptr;

// erste Adresse
  /* Gib mir Speicher ! */
  adr1=(struct adresse *)malloc(sizeof(struct adresse));

  /* Eingabe des Namens */
  printf("Name: "); gets(adr1->name);

  /* Eingabe der Hausnummer */
  printf("Hausnummer: "); gets(s1);
  adr1->hnummer=strtol(s1,&endptr,10);
  if (*endptr!=0) printf("der Rest von help ist : %s\n",endptr);
/* 10 ist die basis, help ist der string, endptr ist der Zeiger auf erste Nichtzahl*/

  /* Speicheradresse des  nachfolgenden Listenelementes ist noch nicht bekannt !*/
  /* wird daher auf Null gesetzt */
  adr1->naechster=NULL;

  anfang=adr1;  /* Nur beim ersten Listenelement */
  vor=adr1;     /* Vorgaengeradresse */


// zweite Adresse
```

```
  adr1=(struct adresse *)malloc(sizeof(struct adresse));
  printf("Name: "); gets(adr1->name);
  adr1->naechster=NULL;

  vor->naechster=adr1;
  vor=adr1;

// dritte Adresse
  adr1=(struct adresse *)malloc(sizeof(struct adresse));
  printf("Name: "); gets(adr1->name);
  adr1->naechster=NULL;

  vor->naechster=adr1;
  vor=adr1;

printf("Ausgabe der Listenelemente \n");
printf("========================== \n");
position=anfang;
do
{
  last=position;
  printf("Name: %s\n",position->name);
  position=position->naechster;
}
while (position!=NULL);

printf("Letztes Element: %s\n",last->name);

}
```

12.2 Aufgaben

1. Durch welche Parameter ist eine Liste eindeutig charakterisiert?
2. Erzeugen Sie eine Literaturdatenbank auf Basis einer Listenstruktur unter Verwendung von *struct*-Variablen bestehend aus mindestens vier Büchern (der klassischen Literatur?). Jedes Buch ist durch Titel, Autor, Auflage und Preis eindeutig charakterisiert.
3. Versuchen Sie die Eingabe aller Listenelemente mit einer Wiederholungsanweisung zu realisisieren! Bei Eingabe des Buchstaben `x` soll das programm beendet werden.
4. Geben Sie die Liste auf dem Bildschirm aus!
5. Fügen Sie nach dem zweiten Listenelemnt ein zusätzliches Listenelement ein!

6. Löschen Sie das vierte Listenelement!
7. Suchen Sie in allen Listenelementen nach der Buchstabenkombination (`er`!
8. Wie bestimmen Sie die Anzahl der Listenelemente?
9. Fügen Sie an das Ende der Liste ein zusätzliches Listenelement an! Wie können Sie das Ende einer Liste bestimmen?